GK: Implementing Full First Order Default Logic for Commonsense Reasoning (System Description)

Tanel Tammet, Dirk Draheim, Priit Järv

Tallinn University of Technology, Tallinn, Estonia

Wikipedia

In artificial intelligence (AI), commonsense reasoning is a human-like ability to make presumptions about the type and essence of ordinary situations humans encounter every day.

These assumptions include judgments about the nature of physical objects, taxonomic properties, and peoples' intentions.

Our project for hybrid ML + Automated Reasoning

- A commonsense reasoner GK https://logictools.org/gk/ , built on top of our resolution-based FOL prover GKC.

- A semantic parser converting English to logic, built on top of the Stanza parser generating the Universal Dependencies graph.

- Adapting and integrating existing commonsense knowledge bases for our purposes: Quasimodo, ConceptNet, Wordnet and more.

Automated reasoning needs a wealth of capabilities for commonsense A.I. to be usable for a hybrid AR + ML system

- Find answers, i.e. substitutions
- Survive contradictions
- Estimate confidences
- Handle defeasible reasoning
- Handle analogy-based reasoning
- Manage context
- Integrate with NLP systems
- Integrate with knowledge bases

Enhance the existing automated reasoning systems towards these!

Automated reasoning needs a wealth of capabilities for commonsense A.I. to be usable for a hybrid AR + ML system

- Find answers, i.e. substitutions
- Survive contradictions
- Estimate confidences
- Handle defeasible reasoning
- Handle analogy-based reasoning
- Manage context
- Integrate with NLP systems
- Integrate with knowledge bases

Implemented in the GK reasoner

Work in progress

A small example of NL input / output

Birds can fly. Penguins are birds. Penguins cannot fly.
Patrick is a penguin and Billie is a bird.
Mickey is perhaps a bird. Theo is possibly a penguin.

Who can fly?
Billie and maybe Mickey.
Who cannot fly?
Patrick and likely Theo.
Patrick can fly?
No.
Billie can swim?
Unknown.

The standard example of default logic

penguin(p)

bird(b)

$\forall x.\ \text{penguin}(x) \Rightarrow \text{bird}(x)$

$\forall x.\ \text{penguin}(x) \Rightarrow \neg\ \text{fly}(x)$

And the default rule $\text{bird}(x) : \text{fly}(x) \vdash \text{fly}(x)$

we encode this as $\neg\ \text{bird}(x) \ \vee\ \text{block}(0, \text{neg}(\text{fly}(x))) \ \vee\ \text{fly}(x)$

The standard example of default logic

Negated goal: fly(x)
The prover runs for N seconds, deriving clauses, including some answer clauses:
….
ans(b)
…
ans(p) V block(0,neg(fly(p)))
….

Now check whether –fly(p) is provable, i.e. fly(p) gives a contradiction:
it does, without any blockers in the result.

Hence b is the sole answer.

Finite domains vs a non-omniscient system

Since FOL is not decidable, it is in general impossible to guarantee that some blocker literal is not derivable.

Hence, the standard approach for handling default logic has been creating a large ground instance KBg of the KB, and then performing decidable propositional reasoning on the KBg. This is what ASP systems do.

Finite domains vs a non-omniscient system

Our approach is to avoid grounding and accept that the system is not logically omniscient:

- Justification checking of blockers (i.e. exceptions) is delayed until a first-order proof is found; after that recursively deepening checks are performed with diminishing time limits.
- GK first produces a potentially large number of different candidate proofs and then enters a recursive checking phase.
- The results produced by GK depend on the time limits and are not stable in the general case.

Set time limit L and a trust threshold confidence Ct

1. Calculate positive evidence with a total confidence Cp
   - Collect Np proofs for the query during L/4 – ε time
   - Attempt to recursively invalidate each of 1 … Np proofs giving (L/4 – ε) / Np time

     …

   A proof is invalidated if some blocker is proved with a confidence over Ct

2. Calculate negative evidence with a total confidence Cn: symmetric to the procedure above

3. Combine positive / negative evidence
   - If Cp > Cn, give a validated combined proof of the query with the confidence Cp – Cn,
   - else a validated combined proof of the negated query with the confidence Cn - Cp

Priorities: an ordering of default rules

The concept of priorities for default rules has been well investigated, with several mechanisms proposed. Defaults are typically ordered by specificity:

Default rules for a more specific class of objects should take priority over rules for more general classes.

For example,

- Birds (who typically do fly) are physical objects.
- Physical objects typically do not fly.
- Hence we have contradictory default rules.
- Since birds are a subset of physical objects, the flying rule of birds should have a higher priority than the non-flying rule of physical objects.

Encoding priorities

- We encode priority information as a first argument of the blocker literal, offering several ways to determine priority: an integer, a taxonomy class number, a string in a taxonomy or a combination of these with an integer.

- For automatically using specificity we employ taxonomy classes: a class has a higher priority than those above it on the taxonomy branch.

- To enable more fine-grained priorities, an integer can be added to the term like $("bird", 2) generating a lexicographic order.

Fine-grained priorities are important

Levels like the “Physical objects cannot fly”, “Birds can fly” and “Penguins cannot fly” are solved using taxonomy-based priorities.

But consider “Birds can fly” and “Baby birds cannot fly”: there is no class taxonomy involved.

We need to be able to attach a potentially arbitrarily deep list of lexicographically comparable priority information to each exception. Consider “Birds can eat meat” and “Baby birds cannot eat raw meat”.

The main modification to the FOL reasoner

Limit subsumption and simplifications with the requirement:

A triple T1 = <A1, c1, L1> consisting of a clause A1, confidence c1 and a dependency list L1 subsumes a triple T2 = <A2, c2, L2> if and only if

- A1 subsumes A2 in the standard sense,
- For blockers *block(s1,t)* in A1 subsuming a blocker *block(s2,g)* in A2, *s2* is not stronger than *s1*.
- If L2 contains a goal clause, L1 also contains a goal clause
- c1 ≥ c2 and
- L1 is a (non-strict) subset of L2

Current ASP limits

Consider a standard example in ASP syntax

```
bird(b1).
penguin(p1).
bird(X) :- penguin(X).
flies(X) :- bird(X), not -flies(X).
-flies(X) :- penguin(X).
```

Ask `flies(b1)` and `flies(p1).`

All default-solving ASP systems can solve these very quickly.

Adding function symbols

When we add the rules

```
bird(father(X)) :- bird(X).
penguin(father(X)) :- penguin(X).
```

Then no ASP systems we tried terminate on the previous queries.

s(CASP) terminates, when the previous formulation is modified as

```
flies(X) :- bird(X), not abs(X).
abs(X) :- penguin(X).
```

Adding transitivity

When we instead add facts and rules

```
father(b1,b2).
father(p1,p2).
...
father(bN-1,bN).
father(pN-1,pN).
ancestor(X,Y):- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
```

for a large N, s(CASP) does not terminate and two other systems become slow for `flies(b1)`: ca 8 seconds for N = 500 and ca 1 minute for N = 1000. GK solves the same question with N = 1000 under half a second and with N = 100000 under three seconds

A simple multi-level example with GK

```
[
[["flyingpenguin", "?:X1"], "=>" ,["penguin", "?:X1"]],
[["penguin", "?:X1"], "=>" ,["bird", "?:X1"]],
[["bird", "?:X1"], "=>" ,["organism", "?:X1"]],
[["organism", "?:X1"], "=>" ,["object", "?:X1"]],

[["penguin", "?:X1"], "<=>" ,["penguin", ["father","?:X1"]]],
[["bird", "?:X1"], "<=>" ,["bird", ["father","?:X1"]]],
[["flyingpenguin", "?:X1"], "<=>" ,["flyingpenguin", ["father","?:X1"]]],

[["flyingpenguin", "?:X1"], "=>" ,[["fly", "?:X1"],"|",["$block",["$","penguin",3],["$not",["fly", "?:X1"]]]]],
[["penguin", "?:X1"], "=>" ,[["-fly", "?:X1"],"|",["$block",["$","penguin",2],["fly", "?:X1"]]]],
[["bird", "?:X1"], "=>" ,[["fly", "?:X1"],"|",["$block",["$","bird"],["$not",["fly", "?:X1"]]]]],
[["organism", "?:X1"], "=>" ,[["-fly", "?:X1"],"|",["$block",["$","organism"],["fly", "?:X1"]]]],

["flyingpenguin","fp"],
["penguin","p"],
["bird","b"],
["organism","o"],

//{"@question": ["fly",["father","?:X"]]}
//{"@question": ["fly","?:X"]}
{"@question": ["fly",["father", ["father","p"]]]}
]
```

Proof output


```
{"result": "answer found",
"answers": [
{
"answer": false,
"blockers": [["$block",["$","penguin",2],["fly",["father",["father","p"]]]]],
"confidence": 1,
"negative proof":
[
[1,     12, ["in", "frm_9", "axiom", 1, []], [["$block",["$","penguin",2],["fly","?:X"]], ["-
penguin","?:X"], ["-fly","?:X"]]],
[2,      5, ["in", "frm_5", "axiom", 1, []], [["penguin",["father","?:X"]], ["-penguin","?:X"]]],
[3,     16, ["in", "frm_13", "axiom", 1, []], [["penguin","p"]]],
[4,  47461, ["mp", [2,1], 3, "fromaxiom", 1, [5,16]], [["penguin",["father","p"]]]],
[5,      5, ["in", "frm_5", "axiom", 1, []], [["penguin",["father","?:X"]], ["-penguin","?:X"]]],
[6,  47465, ["mp", 4, [5,1], "fromaxiom", 1, [5,16]], [["penguin",["father",["father","p"]]]]],
[7,     20, ["in", "$auto_negated_question", "goal", 1, []], [["fly",["father",["father","p"]]]]],
[8,  47540, ["mp", [1,1], 6, 7, "fromgoal", 1, [12,5,16]],
[["$block",["$","penguin",2],["fly",["father",["father","p"]]]]]
]}
]}
```

Using Quasimodo with GK

- Quasimodo is a large commonsense KB enhanced with numeric confidences, built by a group in Max Planck.
- Size is ca 200 megabytes, contains over million rules/facts.
- We have converted Quasimodo and added default conditions to most rules.
- Used by reading the whole KB into shared memory along with the Wordnet taxonomy classes.
- GK is able to solve simple examples using default logic and the whole Quasimodo
- It can also use a large converted subset of ConceptNet along with Quasimodo

Work to do

Completeness questions:

- Define sensible completeness criterias
- Prove the completeness of the algorithm for several subclasses

Employ machine learning and simpler statistics:

- Similarity reasoning using word similarity measures and exceptions
- Clause selection using NLP semantics and co-occurrence embeddings
- Use machine learning to speed up search
- Improve NLP parsing
- Modify and integrate existing commonsense KBs

Summary

Non-grounded full FOL search with delayed justification checking using hierarchical time limits works OK for nontrivial examples of default logic.

Fine-grained lexicographically ordered priorities of defaults are necessary for commonsense reasoning.

Numeric confidences are useful in cases where the default rules are not sufficient.

ML should be used to help us parse NLP, build up a KB and guide the search from a large commonsense KB. For explainability we need to get a proof tree in the end.